Software Specification and Design Document

for

Cribb

An iOS App for Reviewing Off-Campus Housing

Submitted by

Kevin Fu
Daniel Li
Silin Chen
Kathleen Monje
Gianna Cacciatore

in partial fulfillment of the requirements for

CIS 454 — Software Implementation

30 April 2019

Table of Contents

1.0 Introduction

1.1 *Background & Purpose*

Cribb, in a few words, is a system for rating and reviewing off-campus housing among tenants. Purely developed for the iOS environment, the creators of this software project recognized the need for a better way to go about the housing search, especially for college students such as themselves who look for temporary housing every year. Most, if not all, off-campus housing options advertise themselves as the *best* option out there, which can be biased and tough for students in the decision-making process for choosing the *truly best* one. The motivations of this app are primarily aimed at allowing undergraduate and postgraduate students to post and share clean, honest, unbiased reviews on properties they've lived in around campus, as a collective resource around campus — created by, for, and among students only.

This Software Specification and Design Document had been written by the same people behind the development of *Cribb* to outline the system describing how the app operates, and the structuring of data and processes within it, as well as how the user interacts with this system.

1.2 *Scope & Limitations*

Although the main goal of *Cribb* reads: "to allow undergraduate and postgraduate students to post and share *clean, honest, unbiased reviews* on properties they've lived in around campus," still, prevention against abuse and exploitation cannot possibly be completely guaranteed in any system, even with software technology and security. For example, there is no way to authenticate that a user posting a review on a property is not in any way affiliated with the owner, landlord, or leasing company behind that particular property — authentication can only be taken so far. As such, the authentication process in *Cribb* that *is* implemented involves a guard that simply verifies whether a user signs up with a "valid" Syracuse University email (i.e. *user@syr.edu*) as provided by the University to students, faculty, and staff — otherwise, the user isn't able to sign up for a *Cribb* account. For the purposes of this project, this guard was implemented purely by checking that a user's sign-up email is of the form *user@syr.edu*, since third-party app developers do not have access to the official Syracuse University email database. In a potential upscaled version of the app, this can be taken a step further through an email authentication function as provided by *Firebase*, the main real-time storage and database service behind *Cribb*, *in addition to* the *user@syr.edu* format verification described above. On another note, the nature of an open platform such as *Cribb* implies the risks that come with the right to freedom of speech — even of fully verified users. This risk is one that is unavoidable.

In its current version, there are two types of *Cribb* users: **the admin user**, under which only the five credited developers of this app fall, and **the general user**, pertaining to any other user who wishes to use the app and its features.

One limitation is in the authority of even admin users in terms of imposing authentication and security throughout the app and its data. For example, any user is able to add a new property, or "cribb," to the database, if the user wishes to post a review on a

property that is not already listed in the app. Given the nature of this use case, *any user can create a listing for any property, without a proper way of verifying that the information they supply is correct (i.e. landlord's official name, rent price)*. In the case that some wrong piece of information is added to the database, the only way to correct this involves another user submitting a report form, which can only be seen by an admin user. An admin user can then choose to ignore the report, or update the database with the accurate information. However, there might still be no way to verify that even this new information is correct. This poses a limitation as to the extent to which an admin user can monitor the app. A potential future resolution to this can lie in authenticating all information from any user *before* a new property is officially added to the database.

Another limitation is in that although *Cribb* is currently catered towards Syracuse University students and, respectively, housing and apartments around the SU campus area, the developers have not implemented a restriction on how far out a property can be to be considered off-campus housing. As it is, any *valid* address is considered a valid property on *Cribb*, no matter the location, city, state, or country. Again, in an upscaled version of the app, restrictions can easily be set in place, and *Cribb* can expand its horizons to cater to other campuses and schools as well.

As it stands, *Cribb* is currently a completely third-party app, featuring no affiliations with any of the owners, landlords, and leasing companies referenced in the app whatsoever. This is to preserve the unbiased nature of reviews throughout this platform. Future versions can possibly feature ways to display marketing and contact information, exactly as advertised by these companies, alongside the ideally unbiased reviews posted by *Cribb* users, as well.

2.0 Software Requirements Specification

The following sections establish the different software requirements specifications that govern the system that *Cribb* was built to provide to its users. Because a software system design is a living structure, these diagrams were finalized by the authors of this document — the developers of *Cribb* — after the development process, although the system's design was initially established and pre-finalized prior to the creation of the app.

2.1 *Non-Functional Requirements*

2.1.1 Operational Requirements

1. The system will operate only in the iOS environment.
2. The system will also require a valid Internet connection to run on an iOS device.
3. The system will store all data and information on the cloud database.
4. The system will back up through the cloud database in real-time, with a refresh period of about 2 seconds or less.

2.1.2 Performance Requirements

1. The software will process modifications made to the database in 2 seconds or less.

2.1.3 Security Requirements

1. Users must register with their Syracuse University email to verify that they are a current or former SU student.
2. Only the five credited developers of this app, the names of whom are mentioned on the cover page of this document, are considered admin users who are then authorized to perform use cases only admin users are able to do so:
    a. view and delete submitted user reports
    b. delete user reviews
    c. modify and delete property listings

2.1.4 Cultural & Political Requirements

1. No special cultural and political requirements are anticipated at this time.

2.2 *Functional Requirements*

2.2.1 User Account Management

1. A user signs up for a new account.
2. A user logs in to an existing account.
3. A user logs out of his or her account.
4. A user opts to post a new review or modify an existing review to show anonymously (or not).

2.2.2 User Review Management

1. A user writes a new review on an existing property.
2. A user views an existing review written by another user on an existing property.
3. A user modifies an existing review he/she wrote on an existing property.
4. A user deletes an existing review he/she wrote on an existing property.
5. An admin user deletes an existing review written by another user on an existing property.

2.2.3 Property Listing Management

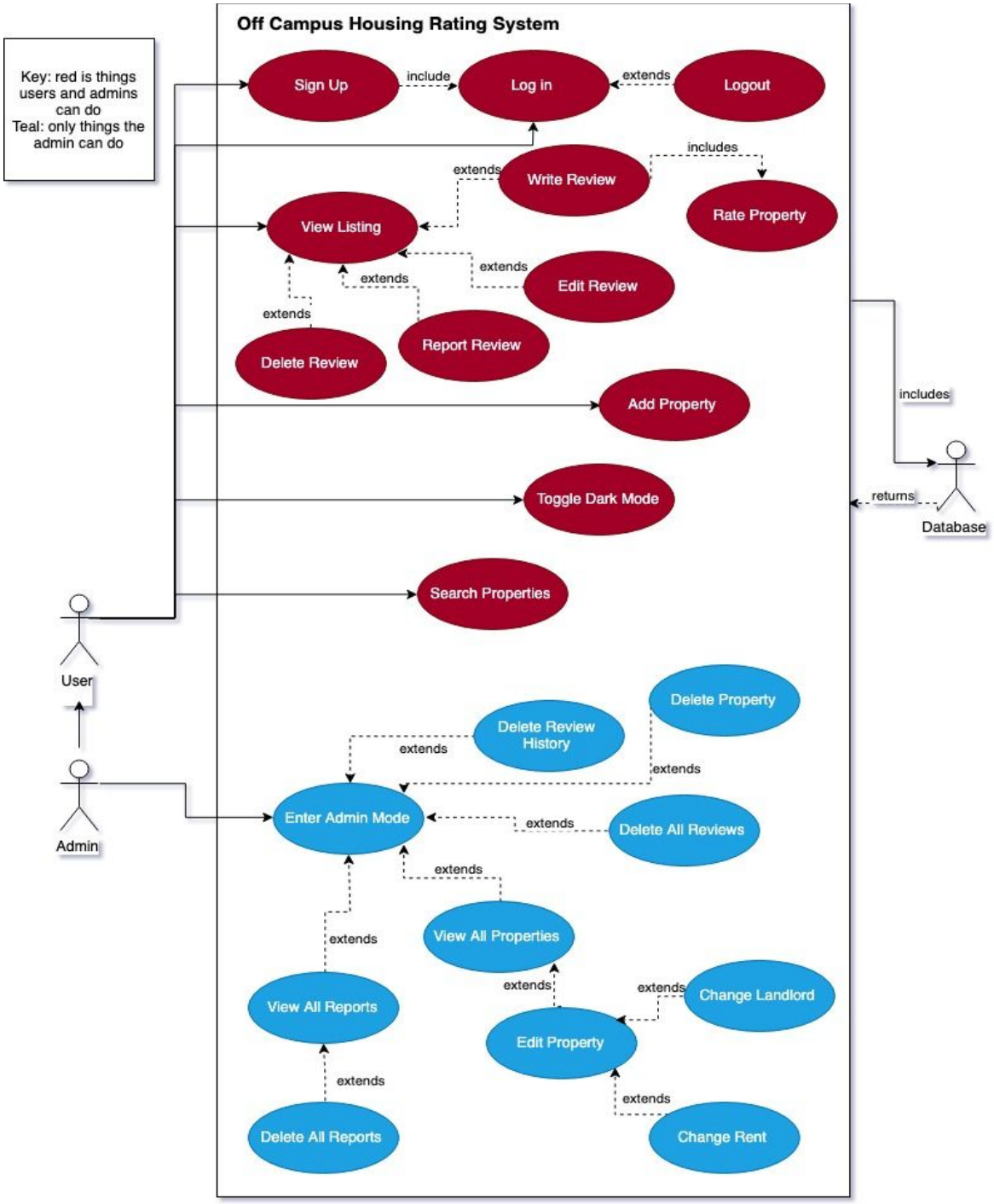
1. A user creates a new property listing to be added to the database.
2. A user performs a search for an existing property throughout the database.
3. A user views an existing property listing.
4. An admin user modifies information listed under a property listing and updates the database.
5. An admin user deletes a property listing and removes it from the database.

2.2.4 User Report Management

1. A user writes and submits a report on an existing property.
2. An admin user views an existing user report.
3. An admin user deletes an existing user report and removes it from the database.

2.3 *Use Case Specifications*

2.3.1 General Use Case Diagram



2.3.2 Use Case Descriptions

USE CASE NAME: Add a property	ID: 1	IMPORTANCE LEVEL: High
PRIMARY ACTOR: User	USE-CASE TYPE: Detail, Essential	
STAKEHOLDERS AND INTERESTS: Potential Member — interested in the greater good of others and others future endeavors Admin — make the app more wholesome for users		
BRIEF DESCRIPTION:	This use case allows anyone to potentially .	
TRIGGER:	User expresses their interest in rating a specific property that's not already in the database	
TYPE:	Internal	
RELATIONSHIPS:		
ASSOCIATION:	Other members	
INCLUDE:	Fill in address, rent price, landlord name and contact info.	
EXTEND:	N/A	
GENERALIZATION:	Greater good for others who want to review too	
NORMAL FLOW OF EVENTS: 1. Member expresses interest in adding a property 2. Types in the address fields, rent, and landlord 3. User will be brought to the Google maps where the property will appear on the screen 4. User can then tap into the pin point and will be brought to another page to review property 5. They will be brought to another page if they decide they want to submit a review.		
SUBFLOWS: They decide they only want to add a property but not review.		
ALTERNATE/EXCEPTIONAL FLOWS: ● User may already be adding a property in the database ● User who is adding a property is adding an invalid address. ● Missing fields or wrong field input		

USE CASE NAME: Adding a review	ID: 2	IMPORTANCE LEVEL: High
PRIMARY ACTOR: User	USE-CASE TYPE: Detail, Essential	

STAKEHOLDERS AND INTERESTS:
User — interested in voicing their opinions about a place they have lived before.

BRIEF DESCRIPTION:	This use case allows users to add a review to the listing. There will be an overall rating, location rating, management rating and amenities rating. All of these scores will be out of a 5 star rating. Users will also be able to write a comment or description to let others know about their experience.
TRIGGER:	Member expresses their interest in voicing their opinions.
TYPE:	Internal

RELATIONSHIPS:

ASSOCIATION:	User
INCLUDE:	Database
EXTEND:	Review Listing
GENERALIZATION:	Increase the number of opinions about a particular property to let others make better leasing decisions.

NORMAL FLOW OF EVENTS:

1. User navigates to a particular property of interest
2. Taps an icon to review which will go to a different screen where they can fill out a form to submit review
3. User will be able to interact with a slider for overall review, and tap the number of stars that they want to give for location, management, and amenities. In addition, they will be able to give comments/ description and decide if they want to make the review anonymous.

SUBFLOWS:
User may have already written a review already, this will check the database and autofill form

ALTERNATE/EXCEPTIONAL FLOWS:

- User gets reported for violating terms and conditions for the app. Will not be able to add a review.
- User got reported for writing too many fake reviews.

USE CASE NAME: Search Lising	ID: 3	IMPORTANCE LEVEL: Medium
PRIMARY ACTOR: User	USE-CASE TYPE: Detail, Essential	

STAKEHOLDERS AND INTERESTS:
Users— would like to search through the huge database of properties to find the one that they are looking for.
Admin - easily find a property that needs attention from reports of a user.

BRIEF DESCRIPTION:	Users will be able to run searches on all the listings in the database. They can filter by overall rating, number of reviews, and overall price. Searches will be able to work by address 1, state, city, zip code, prices ..etc...
TRIGGER:	A user wants to find a listing
TYPE:	Internal

RELATIONSHIPS:

ASSOCIATION:	User, Admin
INCLUDE:	Database
EXTEND:	N/A
GENERALIZATION:	interest

NORMAL FLOW OF EVENTS:

1. User gets annoyed by all of the resources in the database
2. User taps the search button, brings them to a table with all the listings which they can perform searches
3. As soon as they type something in the search, autofill results will appear.
4. Users can then select a listing and that will bring them to a dynamic page where they can review the listing.

SUBFLOWS:
No results come up, so they can choose to add the property.

ALTERNATE/EXCEPTIONAL FLOWS:
N/A

USE CASE NAME: Report a Review	ID: 4	IMPORTANCE LEVEL: High
PRIMARY ACTOR: User	USE-CASE TYPE: Detail, Essential	

STAKEHOLDERS AND INTERESTS:
User - interested in reporting something wrong with the page such as fraud, foul language or something buggy with the app.

BRIEF DESCRIPTION:	This use case will allow users to bring awareness of issues that have occured on the app. Once the user has submitted a report, an alert will pop up apologizing to the user for a bad experience on the app. Report will also apply to rent changes and landlord changes.
TRIGGER:	User sees something wrong with the app
TYPE:	Internal

RELATIONSHIPS:

ASSOCIATION:	Users
INCLUDE:	Write to database
EXTEND:	View Listing
GENERALIZATION:	Report: fraud, app bugg, foul language.

NORMAL FLOW OF EVENTS:
1. User sees something in the not so ordinary.
2. Snitch on a particular user or vents about something fishy or wrong on the app.

SUBFLOWS:

ALTERNATE/EXCEPTIONAL FLOWS:
- User reports often (boy cries wolf) and gets blocked by admin preventing the user to continue submitting their report.

USE CASE NAME: Change Landlord	ID: 5	IMPORTANCE LEVEL: Medium
PRIMARY ACTOR: Admin	USE-CASE TYPE: Internal	

STAKEHOLDERS AND INTERESTS:
Admin — Someone informs an admin that a landlord for a particular property has changed.

BRIEF DESCRIPTION:	Someone informs an admin that a particular property landlord has changed. Admins can then go into the app and change the landlord.
TRIGGER:	Landlord change
TYPE:	Internal

RELATIONSHIPS:

ASSOCIATION:	Admin, landlord
INCLUDE:	update database
EXTEND:	Edit Property
GENERALIZATION:	N/A

NORMAL FLOW OF EVENTS:

1. Admin is informed by someone in the report that landlord has changed.
2. Admin will go into that property and be able to change the landlord name.

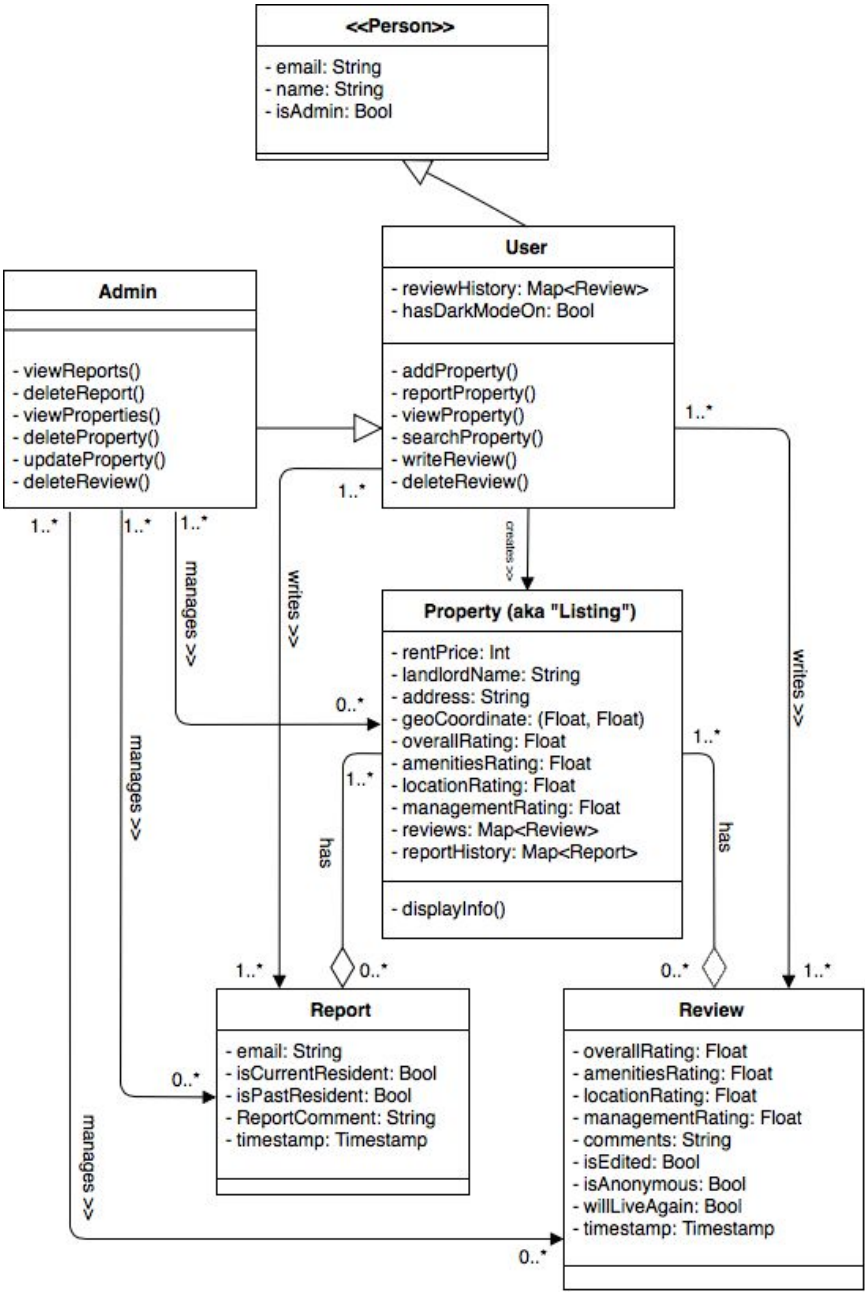
SUBFLOWS:

ALTERNATE/EXCEPTIONAL FLOWS:
Admin realizes that the person claiming that they are the “landlord” is in fact not the real landlord will cause the admin to revert changes or not perform the update in the first place.

2.4 *Class Specifications*

Although the following class specifications do not exactly resemble the source code in structure, the class diagram combines the functional requirements, the use case diagram, and the database design to establish how data is preserved and passed around within the *Cribb* system.

2.4.1 Class Diagram

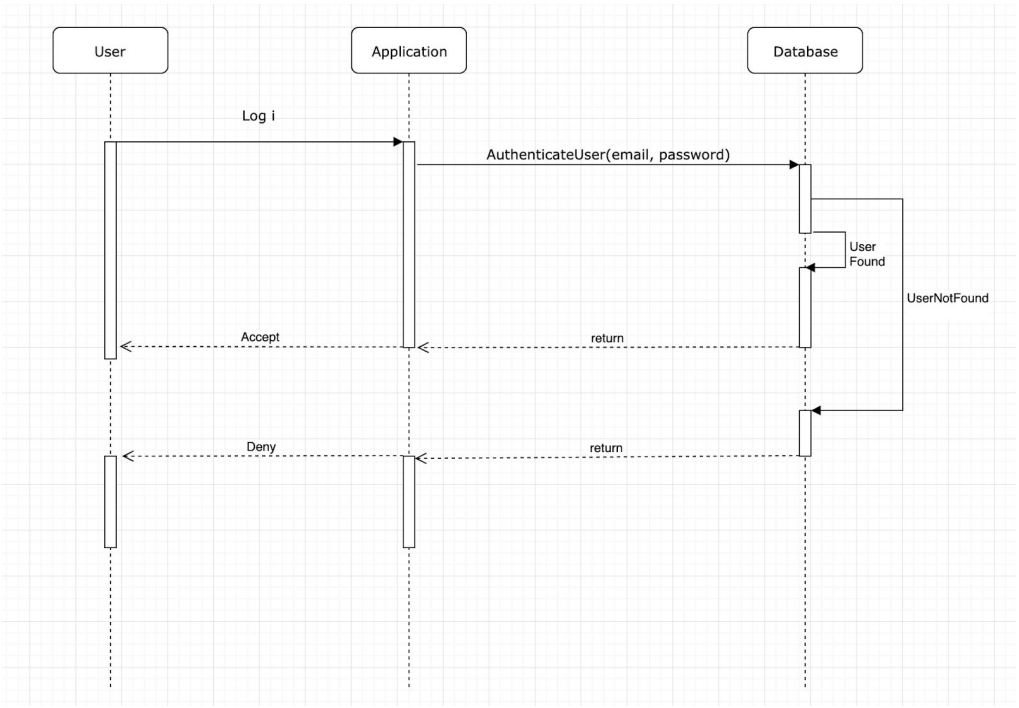


2.4.2 CRUDE Matrix

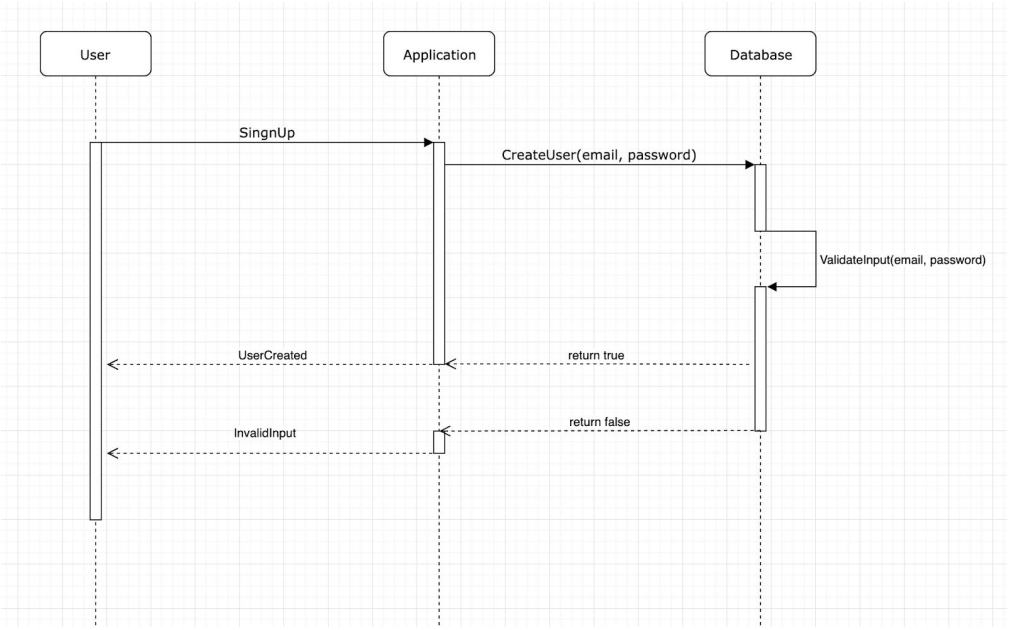
	User	Admin	Property	Review	Report
User			CRE	CRUD	C
Admin			CRUDE	CRUD	CRD
Property				E	
Review					
Report					

2.4.3 Sequence Diagrams

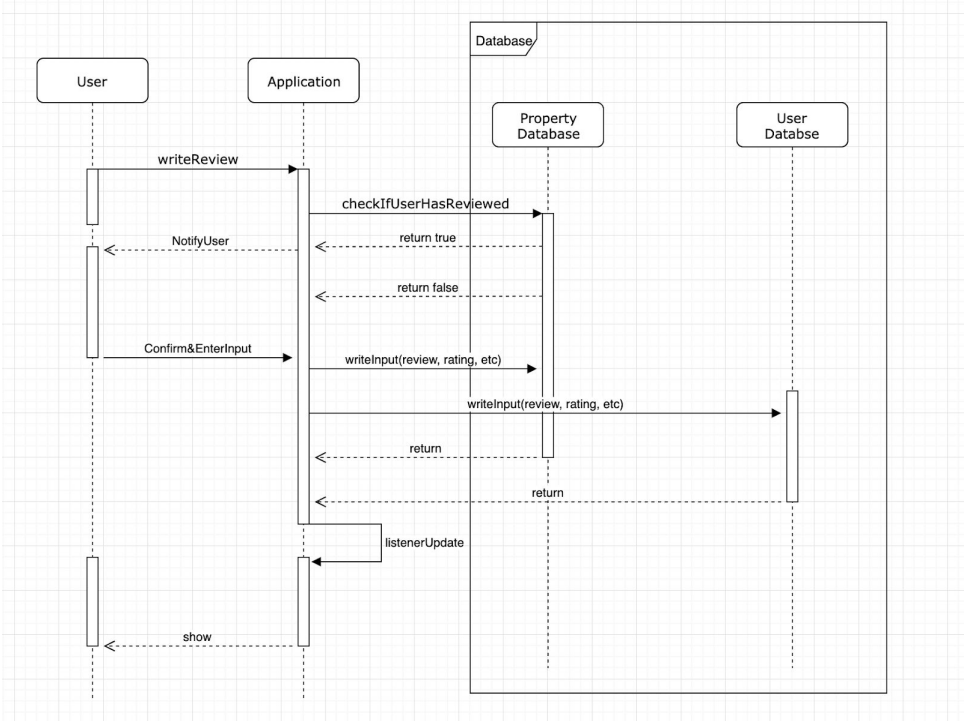
The ***Log In*** Use Case



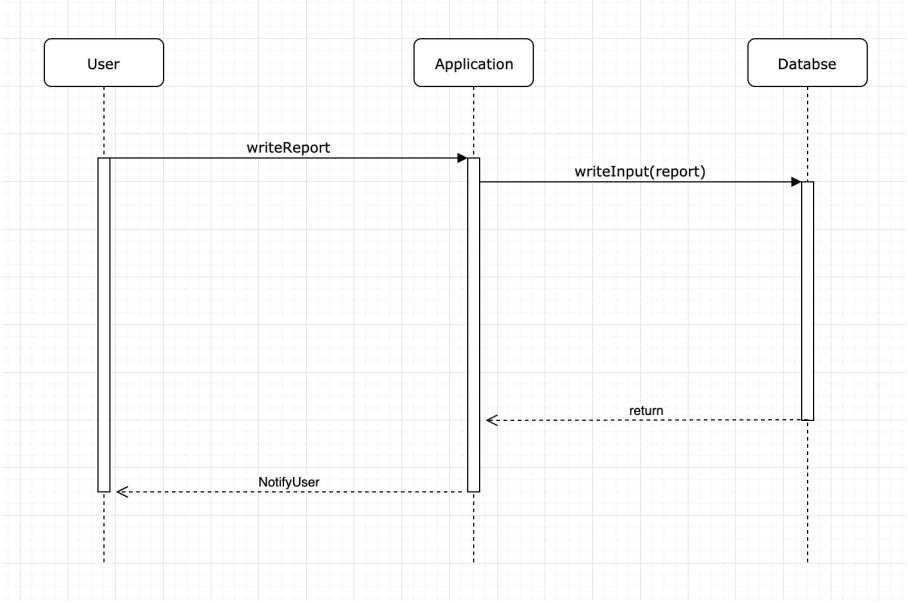
The ***Sign Up*** Use Case



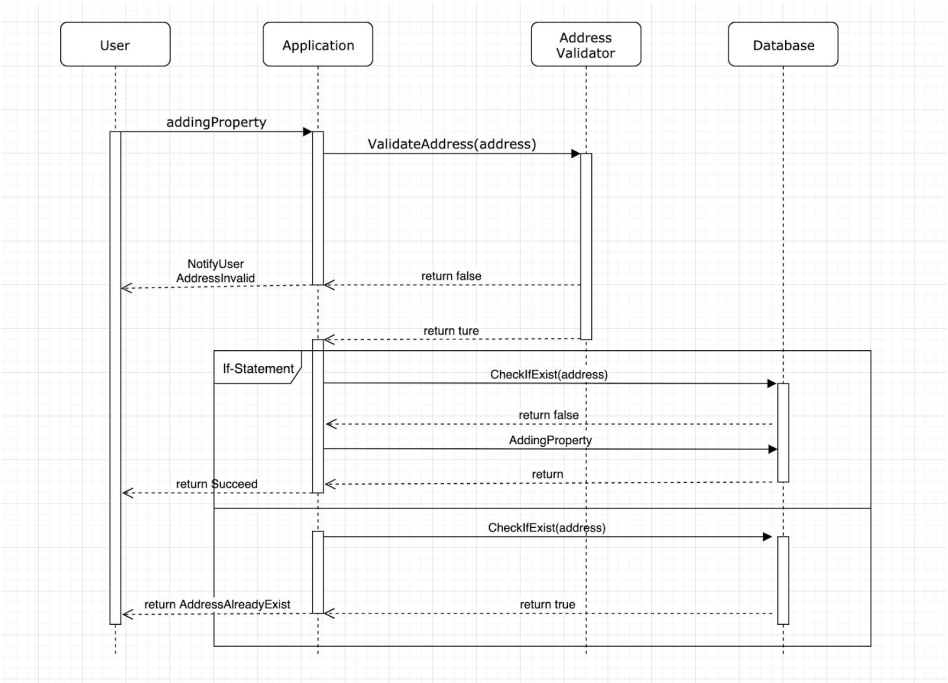
The ***Write Review*** Use Case



The ***Write Report*** Use Case



The ***Adding Property*** Use Case



3.0 Interfaces

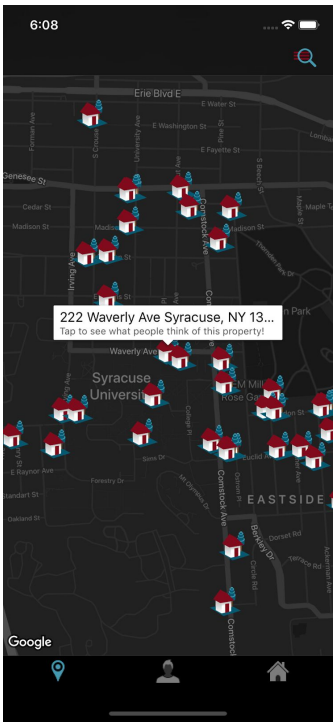
3.1 *Internal Interfaces*

The interface that we designed is minimalistic and simple, allowing the user to navigate the app easily. After the user logs in with a valid SYR email and is authenticated by the database, they are able to use all features of the app. If the user is an admin, they have all the capabilities of a normal user along with the ability to go into admin view.

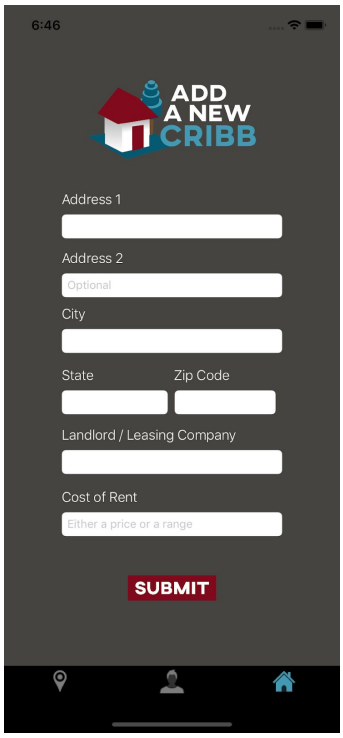
Main Software Portal/User Interface

Splash Page

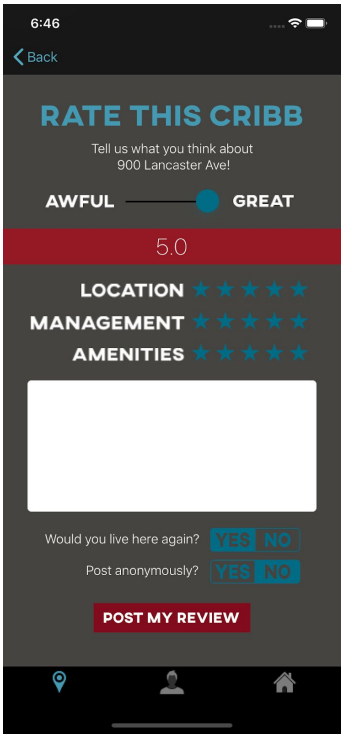
Login Page



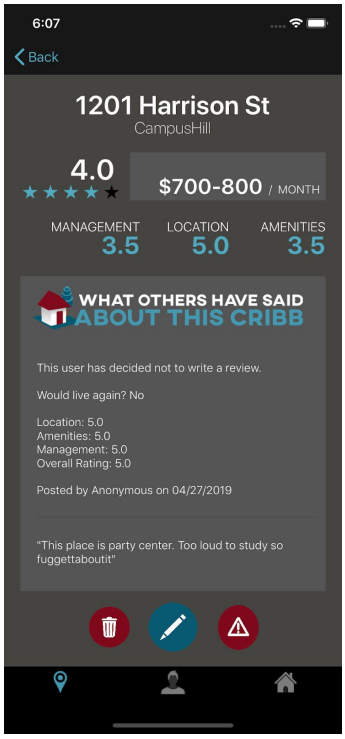
Map View/Main View



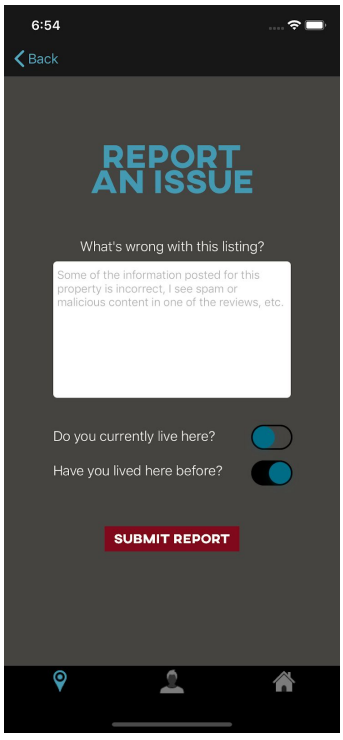
Add new listing



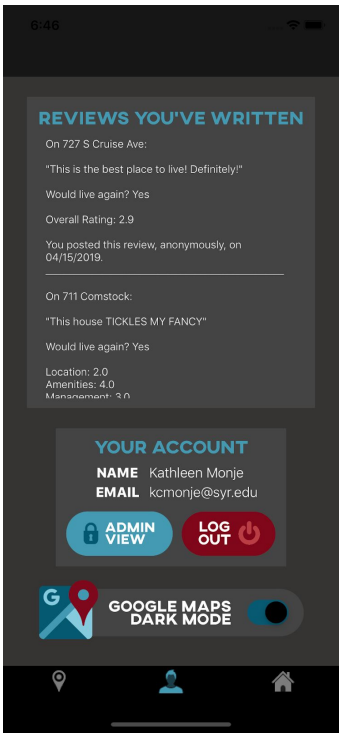
Add a new rating



Viewing a listing



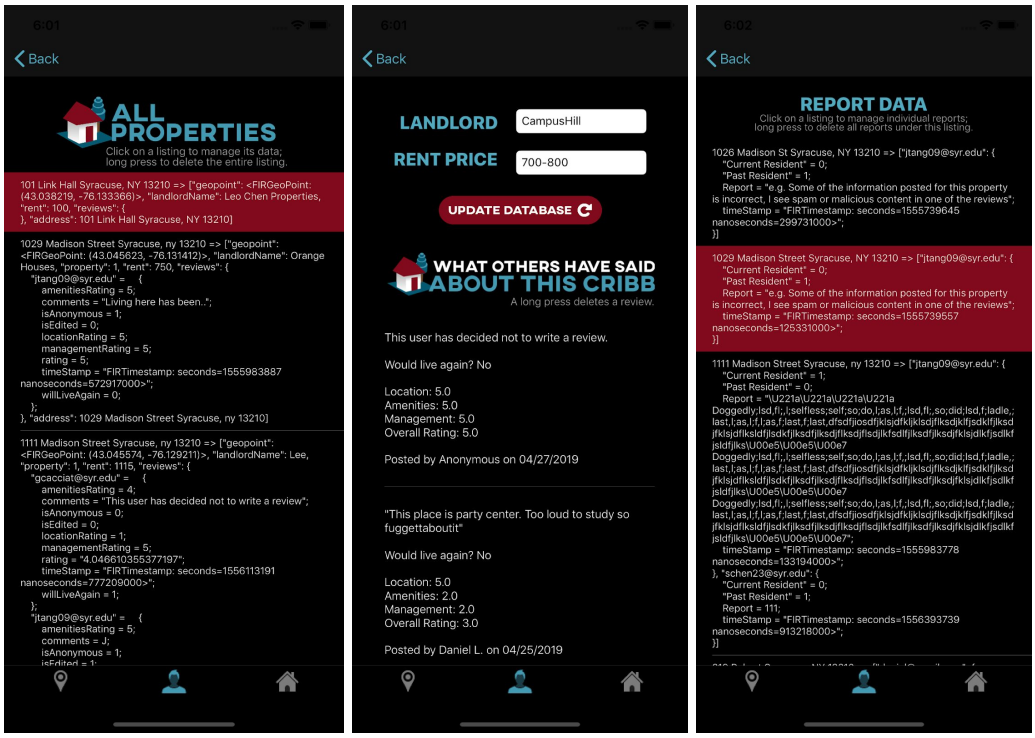
Report an issue



User Profile

Searching for a listing

Admin View



View all properties | Edit a Property | View all reports

3.2 *External Interfaces*

Software Access

The user and admin can access the app through the login page and is granted access once they are authenticated. It's accessed through an app icon on the users phone.

Software Features

The user is able to access features such as adding a new property, viewing and rating a property, editing or deleting their reviews, reporting issues, and searching for properties. Admins have additional features such as viewing all properties and reports as well as deleting reviews and properties.

Software Input

The following data is required from the user: username, user password, and sensitive user information.

Software Output

The software will have various outputs depending on the user including pop-ups for messages and verifications of actions.

4.0 Database Design

4.1 *Database Tables*

The system utilizes five database tables structured as collections — one for:

1. reports,
2. users and review histories, and
3. listings and reviews.

While not itself a collection, the review history database is connected to the user database as another field within the *Users* collection, storing reviews written by each user and associating them with that particular account. The listings and review databases work the same way with the key difference being that the reviews database holds all the reviews for a particular listing.

Attributes and data types for each of these are shown below.

4.1.1 Report Database

REPORTS					
PROPERTY ADDRESS	**USER EMAIL**	**CURRENT RESIDENCE**	**PAST RESIDENCE**	**REPORTED ISSUE**	**TIME STAMP**
(string)	*(string)*	*(boolean)*	*(boolean)*	*(string)*	*(time/date)*

4.1.2 User Database

USERS					
EMAIL ADDRESS	**ADMIN STATUS**	**DARK MODE**	**FIRST NAME**	**LAST NAME**	**REVIEW HISTORY**
(string)	*(boolean)*	*(boolean)*	*(text)*	*(text)*	*(database)*

4.1.2 Review History Database

REVIEW HISTORY									
PROPERTY ADDRESS	**IS ANONY-MOUS**	**HAS BEEN MODIFIED**	**TIME STAMP**	**COMMENT**	**WOULD LIVE AGAIN**	**OVER-ALL RATING**	**MANAGE-MENT RATING**	**AMENI-TIES RATING**	**LOCA-TION RATING**
(string)	*(boolean)*	*(boolean)*	*(time/date)*	*(string)*	*(boolean)*	*(double)*	*(integer)*	*(integer)*	*(integer)*

4.1.3 Listings Database

LISTINGS					
USER EMAIL	**PROPERTY ADDRESS**	**LOCATION**	**LANDLORD**	**RENT**	**REVIEWS**
(string)	*(string)*	*(geopoint coordinates)*	*(string)*	*(string)*	*(database)*

4.1.3 Review Database

REVIEWS									
PROPERTY ADDRESS	**IS ANONY-MOUS**	**HAS BEEN MODIFIED**	**TIME STAMP**	**COMMENT**	**WOULD LIVE AGAIN**	**OVER-ALL RATING**	**MANAGE-MENT RATING**	**AMENI-TIES RATING**	**LOCA-TION RATING**
(string)	*(boolean)*	*(boolean)*	*(time/date)*	*(string)*	*(boolean)*	*(double)*	*(integer)*	*(integer)*	*(integer)*

4.2 *Database Structuring*

Cribb's database is built using *Firestore*, *Google Firebase*'s latest data structure. It is an object-relational structure, because as such a structure significantly suits our needs for this application. Everything within this database can be visualized and accessed quickly. It also allow us to create multiple different collections which we will use to store 3 main different types of data: ***Users***, ***Reports***, and ***Listings***. Each time a user is registered, a new user document will be created just for this user under the *Users* collection. All of the users account details will be stored in their document. Subsequently, every property has its own document under the *Listings* collection, and user reports under the *Reports* collection every time a user submits one. Only admin users have access to the *Reports* collection in the database.

By using such a structure, we can also easily link data between collections. (i.e. when a user writes a new review on a property, this review not only appears under this property's *Review* collection, but also under this user's *Review History* collection) We use various write and read functions to allow collections to communicate with each other. Modifying data in any way within the app triggers a *listener*, which allows for seamless updating of the database, as well as refreshing of the pages on the app.

4.3 *Database Action Restrictions*

Inside of the Reports collection, only the admin will be able to view the contents of that database. This will ensure the full confidentially of users who use our application. All users will be able to write reports as well for any property they desire. As for deleting reports, only admins will be able to delete reviews.

In Users collections, only the user itself is able to read, write and delete in their own document in that collection. They will not be able to read, write or delete any other user in the application. Admins will be the only people who can see everything from user review history, first name, last name.

As far as the listings collection, any user can be able to add a document to the listing. Each document in the database is a property. This function will only be possible with the approval of an admin, once this property is approved user can then see it on the application. All users will be able to see each property and all the reviews that fall under that. Users can only delete the reviews that they have written and admins have the ability to delete all the reviews.

In addition, there will be a limit to how many reviews, reports and add properties that users can write to avoid someone from botting and writing millions of nonsense things on the database.

Ultimately, these security rules will be in place on the firestore security rules. This will ensure that our app does not face a full blown data breach or a hacker trying to delete or spike our database with garbage content.